**MongoDB Notes**

**What is MongoDB: -**

MongoDB is a No SQL database. It is an open-source, cross-platform, document-oriented database written in C++ developed and supported by a company named 10gen.

It provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

**Database:**

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

**Collection:**

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

**Document:**

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB.

|  |  |
| --- | --- |
| **RDBMS** | **MongoDB** |
| Database | Database |
| Table | Collection |
| Tuple/Row | Document |
| column | Field |
| Table Join | Embedded Documents |
| Primary Key | Primary Key (Default key \_id provided by MongoDB itself) |

**What is \_id: -**

**\_id** is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide \_id while inserting the document. If you don’t provide then MongoDB provides a unique id for every document. These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.

**What is the purpose of building MongoDB: -**

**The primary purpose of building MongoDB is:**

* Scalability
* Performance
* High Availability
* Scaling from single server deployments to large, complex multi-site architectures.
* Develop Faster
* Deploy Easier
* Scale Bigger

**Features of MongoDB: -**

These are some important features of MongoDB:

1. Support ad hoc queries

In MongoDB, you can search by field, range query and it also supports regular expression searches.

2. Indexing

You can index any field in a document.

3. Replication

MongoDB supports Master Slave replication.

A master can perform Reads and Writes and a Slave copies data from the master and can only be used for reads or back up (not writes)

4. Duplication of data

MongoDB can run over multiple servers. The data is duplicated to keep the system up and also keep its running condition in case of hardware failure.

5. Load balancing

It has an automatic load balancing configuration because of data placed in shards.

6. Supports map reduce and aggregation tools.

7. Uses [JavaScript](https://www.javatpoint.com/javascript-tutorial) instead of Procedures.

8. It is a schema-less database written in [C++](https://www.javatpoint.com/cpp-tutorial).

9. Provides high performance.

10. Stores files of any size easily without complicating your stack.

11. Easy to administer in the case of failures.

12. It also supports:

JSON data model with dynamic schemas

Auto-sharding for horizontal scalability

Built in replication for high availability

**What is NoSQL Database: -**

Databases can be divided in 3 types:

1. RDBMS (Relational Database Management System)
2. OLAP (Online Analytical Processing)
3. NoSQL (recently developed database)

**NoSQL Database :**

NoSQL Database is used to refer a non-SQL or non-relational database.

It provides a mechanism for storage and retrieval of data other than tabular relations model used in relational databases. NoSQL database doesn't use tables for storing data. It is generally used to store big data and real-time web applications.

**Where MongoDB should be used: -**

* Big and complex data
* Mobile and social infrastructure
* Content management and delivery
* User data management
* Data hub

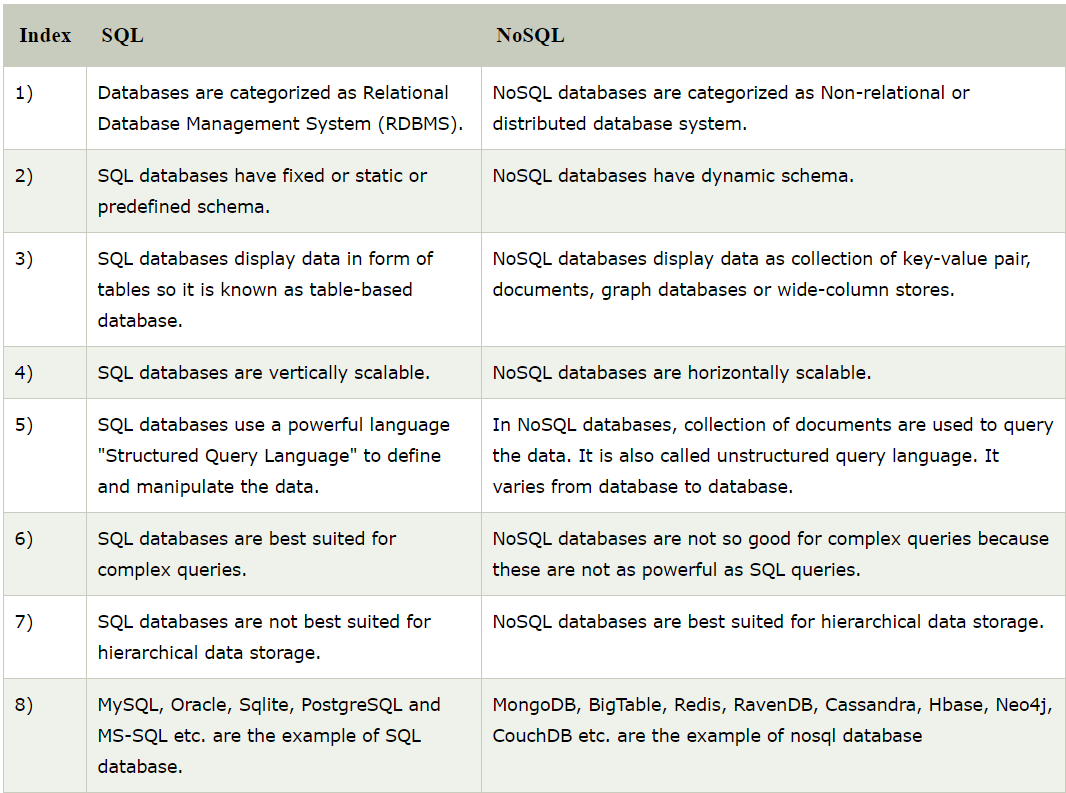
**MongoDB Data Types: -**

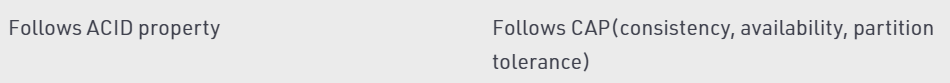
MongoDB supports many datatypes. Some of them are −

* **String** − This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
* **Integer** − This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
* **Boolean** − This type is used to store a boolean (true/ false) value.
* **Double** − This type is used to store floating point values.
* **Min/ Max keys** − This type is used to compare a value against the lowest and highest BSON elements.
* **Arrays** − This type is used to store arrays or list or multiple values into one key.
* **Timestamp** − ctimestamp. This can be handy for recording when a document has been modified or added.
* **Object** − This datatype is used for embedded documents.
* **Null** − This type is used to store a Null value.
* **Symbol** − This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
* **Date**− This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
* **Object ID** − This datatype is used to store the document’s ID.
* **Binary data** − This datatype is used to store binary data.
* **Code** − This datatype is used to store JavaScript code into the document.
* **Regular expression** − This datatype is used to store regular expression.

**Difference Between SQL & Nosql: -**

Following is a list of differences between SQL and NoSQL database



9)

**MongoDB commands**: -

**To show Databases: -**

If you want to check your databases list, use the command **show dbs**.

>show dbs

local 0.78125GB

test 0.23012GB

**The use Command (To Create Database): -**

This command is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

If you want to use a database with name **<mydb>**, then **use DATABASE** statement would be as follows −

>use mydb

switched to db mydb

Your created database (mydb) is not present in list. To display database, you need to insert at least one document into it.

In MongoDB default database is test. If you didn't create any database, then collections will be stored in test database.

**To show Currently Selected Databases: -**

To check your currently selected database, use the command **db**

>db

mydb

## The dropDatabase() Method (To Drop Database): -

MongoDB **db.dropDatabase()** command is used to drop a existing database.

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

If you want to delete new database **<mydb>**, then **dropDatabase()** command would be as follows −

>use mydb

switched to db mydb

>db.dropDatabase()

>{ "dropped" : "mydb", "ok" : 1 }

>

Now check list of databases.

>show dbs

local 0.78125GB

test 0.23012GB

>

**To show Collections: -**

You can check the created collection by using the command **show collections**.

>show collections

mycollection

system.indexes

In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.

>db.tutorialspoint.insert({"name" : "tutorialspoint"}),

WriteResult({ "nInserted" : 1 })

>show collections

mycol

mycollection

system.indexes

tutorialspoint

>

## The createCollection() Method (To Create Collection): -

MongoDB **db.createCollection(name, options)** is used to create collection.

In the command, **name** is name of collection to be created. **Options** is a document and is used to specify configuration of collection.

Basic syntax of **createCollection()** method without options is as follows −

>use test

switched to db test

>db.createCollection("mycollection")

{ "ok" : 1 }

>

## The drop() Method (To Drop Collection): -

MongoDB's **db.collection.drop()** is used to drop a collection from the database.

>db.mycollection.drop()

true

>

Again check the list of collections into database.

>show collections

mycol

system.indexes

tutorialspoint

>

drop() method will return true, if the selected collection is dropped successfully, otherwise it will return false.

## The insertOne() method (To insert only one): -

If you need to insert only one document into a collection you can use this method.

Following example creates a new collection named empDetails and inserts a document using the insertOne() method.

> db.createCollection("empDetails")

{ "ok" : 1 }

> db.empDetails.insertOne(

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Date\_Of\_Birth: "1995-09-26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9848022338"

})

{

"acknowledged" : true,

"insertedId" : ObjectId("5dd62b4070fb13eec3963bea")

}

>

## The insertMany() method (To insert many at once): -

You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

Following example inserts three different documents into the empDetails collection using the insertMany() method.

> db.empDetails.insertMany(

[

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Date\_Of\_Birth: "1995-09-26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9000012345"

},

{

First\_Name: "Rachel",

Last\_Name: "Christopher",

Date\_Of\_Birth: "1990-02-16",

e\_mail: "Rachel\_Christopher.123@gmail.com",

phone: "9000054321"

},

{

First\_Name: "Fathima",

Last\_Name: "Sheik",

Date\_Of\_Birth: "1990-02-16",

e\_mail: "Fathima\_Sheik.123@gmail.com",

phone: "9000054321"

}

]

)

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("5dd631f270fb13eec3963bed"),

ObjectId("5dd631f270fb13eec3963bee"),

ObjectId("5dd631f270fb13eec3963bef")

]

}

>

## The find() Method (To Display all Documents): -

To query data from MongoDB collection, you need to use MongoDB's **find()** method.

**find()** method will display all the documents in a non-structured way.

Following method retrieves all the documents in the collection −

> db.mycol.find()

{ "\_id" : ObjectId("5dd4e2cc0821d3b44607534c"), "title" : "MongoDB Overview", "description" : "MongoDB is no SQL database", "by" : "tutorials point", "url" : "http://www.tutorialspoint.com", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 100 }

{ "\_id" : ObjectId("5dd4e2cc0821d3b44607534d"), "title" : "NoSQL Database", "description" : "NoSQL database doesn't have tables", "by" : "tutorials point", "url" : "http://www.tutorialspoint.com", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 20, "comments" : [ { "user" : "user1", "message" : "My first comment", "dateCreated" : ISODate("2013-12-09T21:05:00Z"), "like" : 0 } ] }

>

## The pretty() Method (To Display in better format): -

To display the results in a formatted way, you can use pretty() method.

Following example retrieves all the documents from the collection named mycol and arranges them in an easy-to-read format.

> db.mycol.find().pretty()

{

"\_id" : ObjectId("5dd4e2cc0821d3b44607534c"),

"title" : "MongoDB Overview",

"description" : "MongoDB is no SQL database",

"by" : "tutorials point",

"url" : "http://www.tutorialspoint.com",

"tags" : [

"mongodb",

"database",

"NoSQL"

],

"likes" : 100

}

{

"\_id" : ObjectId("5dd4e2cc0821d3b44607534d"),

"title" : "NoSQL Database",

"description" : "NoSQL database doesn't have tables",

"by" : "tutorials point",

"url" : "http://www.tutorialspoint.com",

"tags" : [

"mongodb",

"database",

"NoSQL"

],

"likes" : 20, }

## The findOne() method (To Display only one Document): -

Apart from the find() method, there is **findOne()** method, that returns only one document.

### Syntax

>db.COLLECTIONNAME.findOne()

### Example

Following example retrieves the document with title MongoDB Overview.

> db.mycol.findOne({title: "MongoDB Overview"})

{

"\_id" : ObjectId("5dd6542170fb13eec3963bf0"),

"title" : "MongoDB Overview",

"description" : "MongoDB is no SQL database",

"by" : "tutorials point",

"url" : "http://www.tutorialspoint.com",

"tags" : [

"mongodb",

"database",

"NoSQL"

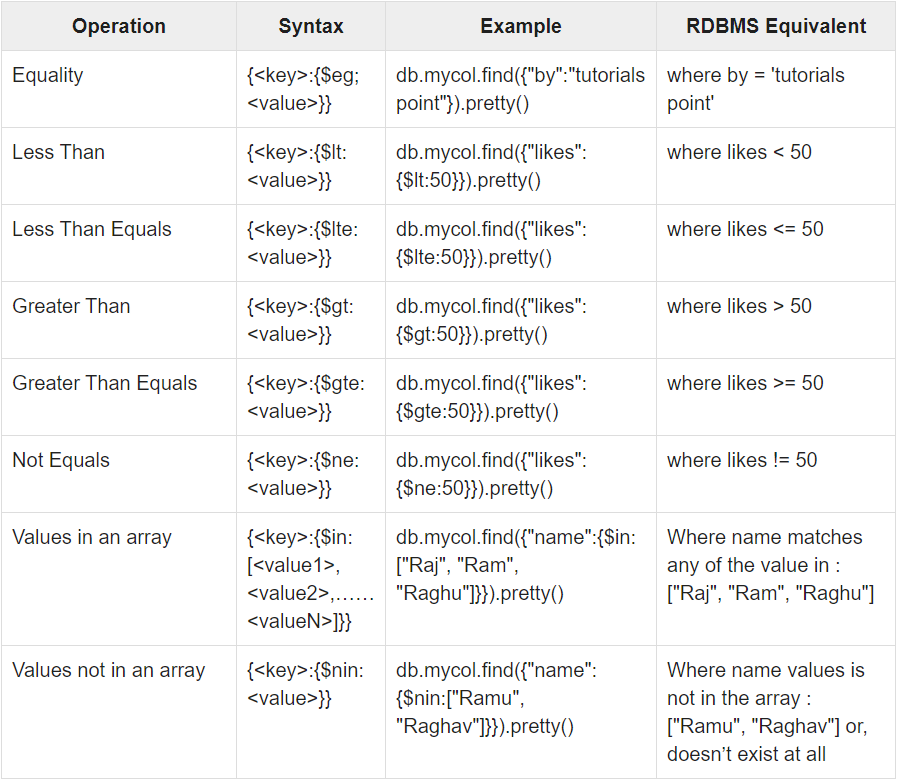
],

"likes" : 100

}

## RDBMS Where Clause Equivalents in MongoDB (Used with Find method):-

To query the document on the basis of some condition, you can use following operations.



## AND in MongoDB: -

### Syntax

To query documents based on the AND condition, you need to use $and keyword. Following is the basic syntax of AND −

>db.mycol.find({ $and: [ {<key1>:<value1>}, { <key2>:<value2>} ] })

### Example

Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

> db.mycol.find({$and:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()

{

"\_id" : ObjectId("5dd4e2cc0821d3b44607534c"),

"title" : "MongoDB Overview",

"description" : "MongoDB is no SQL database",

"by" : "tutorials point",

"url" : "http://www.tutorialspoint.com",

"tags" : [

"mongodb",

"database",

"NoSQL"

],

"likes" : 100

}

>

For the above given example, equivalent where clause will be **' where by = 'tutorials point' AND title = 'MongoDB Overview' '**. You can pass any number of key, value pairs in find clause.

## OR in MongoDB: -

### Syntax

To query documents based on the OR condition, you need to use **$or** keyword. Following is the basic syntax of **OR** −

>db.mycol.find(

{

$or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

### Example

Following example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview'.

>db.mycol.find({$or:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()

{

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "tutorials point",

"url": "http://www.tutorialspoint.com",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

>

## Using AND and OR Together: -

### Example

The following example will show the documents that have likes greater than 10 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent SQL where clause is **'where likes>10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')'**

>db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials point"},

{"title": "MongoDB Overview"}]}).pretty()

{

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "tutorials point",

"url": "http://www.tutorialspoint.com",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

>

## NOR in MongoDB: -

### Syntax

>db.COLLECTION\_NAME.find(

{

$nor: [

{key1: value1}, {key2:value2}

]

}

)

### Example

Assume we have inserted 3 documents in the collection **empDetails** as shown below −

db.empDetails.insertMany(

[

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Age: "26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9000012345"

},

{

First\_Name: "Rachel",

Last\_Name: "Christopher",

Age: "27",

e\_mail: "Rachel\_Christopher.123@gmail.com",

phone: "9000054321"

},

{

First\_Name: "Fathima",

Last\_Name: "Sheik",

Age: "24",

e\_mail: "Fathima\_Sheik.123@gmail.com",

phone: "9000054321"

}

]

)

Following example will retrieve the document(s) whose first name is not "Radhika" and last name is not "Christopher"

> db.empDetails.find(

{

$nor:[

40

{"First\_Name": "Radhika"},

{"Last\_Name": "Christopher"}

]

}

).pretty()

{

"\_id" : ObjectId("5dd631f270fb13eec3963bef"),

"First\_Name" : "Fathima",

"Last\_Name" : "Sheik",

"Age" : "24",

"e\_mail" : "Fathima\_Sheik.123@gmail.com",

"phone" : "9000054321"

}

## NOT in MongoDB: -

### Syntax

To query documents based on the NOT condition, you need to use $not keyword following is the basic syntax of **NOT** −

>db.COLLECTION\_NAME.find(

{

$not: [

{key1: value1}, {key2:value2}

]

}

).pretty()

### Example

Following example will retrieve the document(s) whose age is not greater than 25

> db.empDetails.find( { "Age": { $not: { $gt: "25" } } } )

{

"\_id" : ObjectId("5dd6636870fb13eec3963bf7"),

"First\_Name" : "Fathima",

"Last\_Name" : "Sheik",

"Age" : "24",

"e\_mail" : "Fathima\_Sheik.123@gmail.com",

"phone" : "9000054321"

}

## MongoDB updateOne() method: -

This methods updates a single document which matches the given filter.

### Syntax

The basic syntax of updateOne() method is as follows −

>db.COLLECTION\_NAME.updateOne(<filter>, <update>)

### Example

> db.empDetails.updateOne(

{First\_Name: 'Radhika'},

{ $set: { Age: '30',e\_mail: 'radhika\_newemail@gmail.com'}}

)

{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }

>

## MongoDB updateMany() method: -

The updateMany() method updates all the documents that matches the given filter.

### Syntax

The basic syntax of updateMany() method is as follows −

>db.COLLECTION\_NAME.update(<filter>, <update>)

### Example

> db.empDetails.updateMany(

{Age:{ $gt: "25" }},

{ $set: { Age: '00'}}

)

{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }

You can see the updated values if you retrieve the contents of the document using the find method as shown below −

> db.empDetails.find()

{ "\_id" : ObjectId("5dd6636870fb13eec3963bf5"), "First\_Name" : "Radhika", "Last\_Name" : "Sharma", "Age" : "00", "e\_mail" : "radhika\_newemail@gmail.com", "phone" : "9000012345" }

{ "\_id" : ObjectId("5dd6636870fb13eec3963bf6"), "First\_Name" : "Rachel", "Last\_Name" : "Christopher", "Age" : "00", "e\_mail" : "Rachel\_Christopher.123@gmail.com", "phone" : "9000054321" }

{ "\_id" : ObjectId("5dd6636870fb13eec3963bf7"), "First\_Name" : "Fathima", "Last\_Name" : "Sheik", "Age" : "24", "e\_mail" : "Fathima\_Sheik.123@gmail.com", "phone" : "9000054321" }

>

## MongoDB findOneAndUpdate() method: -

The **findOneAndUpdate()** method updates the values in the existing document.

### Syntax

The basic syntax of **findOneAndUpdate()** method is as follows −

>db.COLLECTION\_NAME.findOneAndUpdate(SELECTIOIN\_CRITERIA, UPDATED\_DATA)

### Example

Assume we have created a collection named empDetails and inserted three documents in it as shown below −

> db.empDetails.insertMany(

[

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Age: "26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9000012345"

},

{

First\_Name: "Rachel",

Last\_Name: "Christopher",

Age: "27",

e\_mail: "Rachel\_Christopher.123@gmail.com",

phone: "9000054321"

},

{

First\_Name: "Fathima",

Last\_Name: "Sheik",

Age: "24",

e\_mail: "Fathima\_Sheik.123@gmail.com",

phone: "9000054321"

}

]

)

Following example updates the age and email values of the document with name 'Radhika'.

> db.empDetails.findOneAndUpdate(

{First\_Name: 'Radhika'},

{ $set: { Age: '30',e\_mail: 'radhika\_newemail@gmail.com'}}

)

{

"\_id" : ObjectId("5dd6636870fb13eec3963bf5"),

"First\_Name" : "Radhika",

"Last\_Name" : "Sharma",

"Age" : "30",

"e\_mail" : "radhika\_newemail@gmail.com",

"phone" : "9000012345"

}

## The remove() Method: -

MongoDB's **remove()** method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

* **deletion criteria** − (Optional) deletion criteria according to documents will be removed.
* **justOne** − (Optional) if set to true or 1, then remove only one document.

### Syntax

Basic syntax of **remove()** method is as follows −

>db.COLLECTION\_NAME.remove(DELLETION\_CRITTERIA)

### Example

Consider the mycol collection has the following data.

{\_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"},

{\_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"},

{\_id : ObjectId("507f191e810c19729de860e3"), title: "Tutorials Point Overview"}

Following example will remove all the documents whose title is 'MongoDB Overview'.

>db.mycol.remove({'title':'MongoDB Overview'})

WriteResult({"nRemoved" : 1})

> db.mycol.find()

{"\_id" : ObjectId("507f191e810c19729de860e2"), "title" : "NoSQL Overview" }

{"\_id" : ObjectId("507f191e810c19729de860e3"), "title" : "Tutorials Point Overview" }

## Remove Only One : -

If there are multiple records and you want to delete only the first record, then set **justOne** parameter in **remove()** method.

>db.COLLECTION\_NAME.remove(DELETION\_CRITERIA,1)

## Remove All Documents : -

If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. **This is equivalent of SQL's truncate command.**

> db.mycol.remove({})

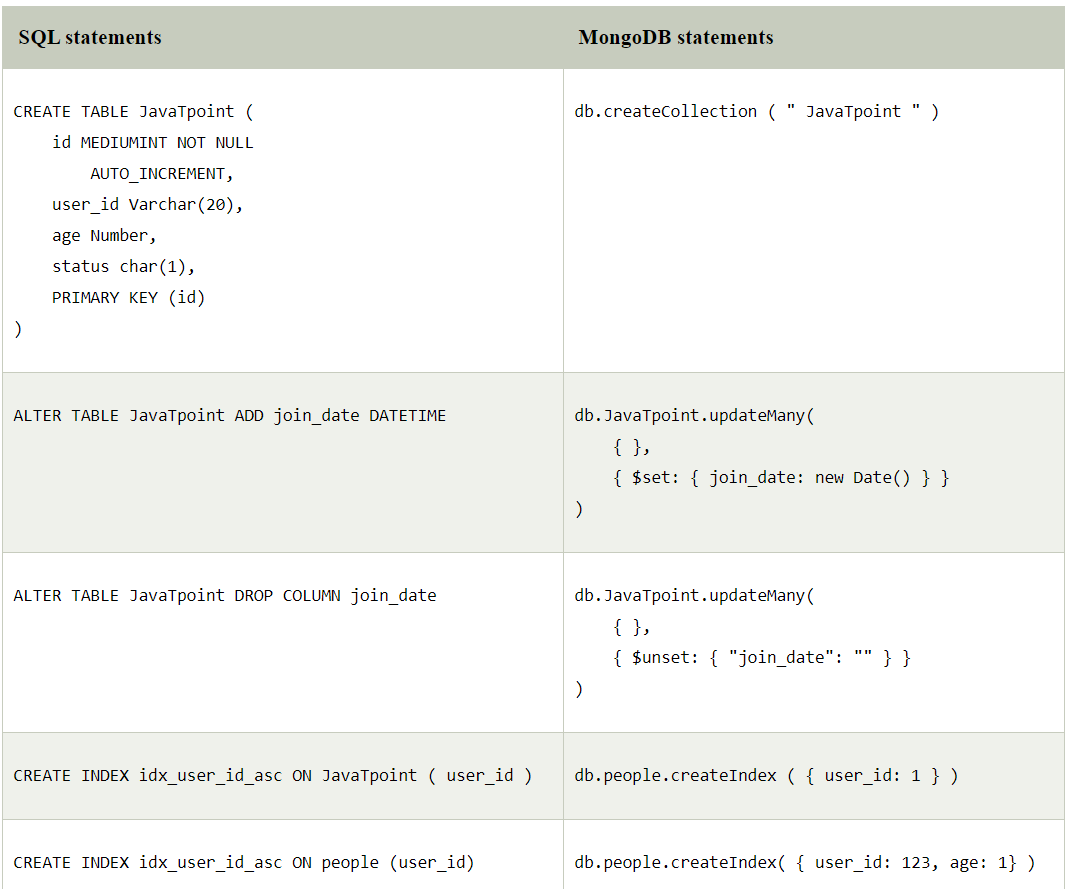
WriteResult({ "nRemoved" : 2 })

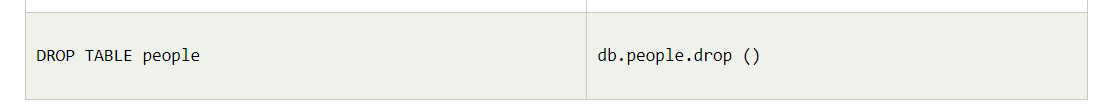
> db.mycol.find()

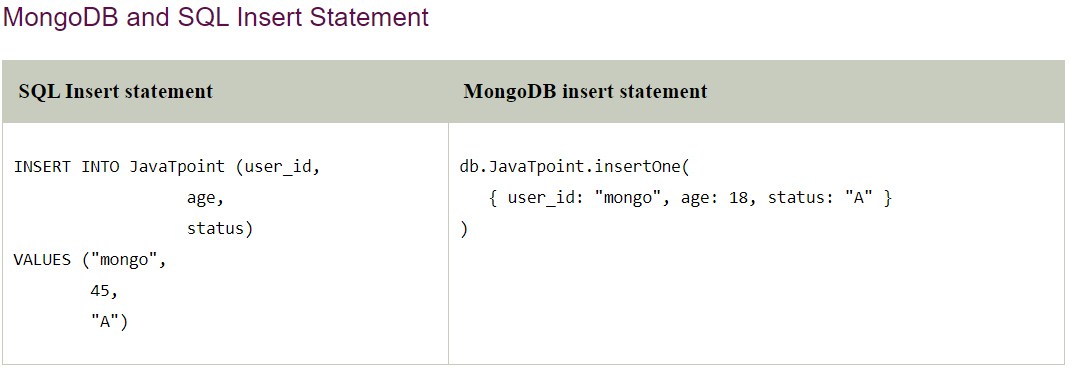
>

**SQL VS MongoDB Commands :-**

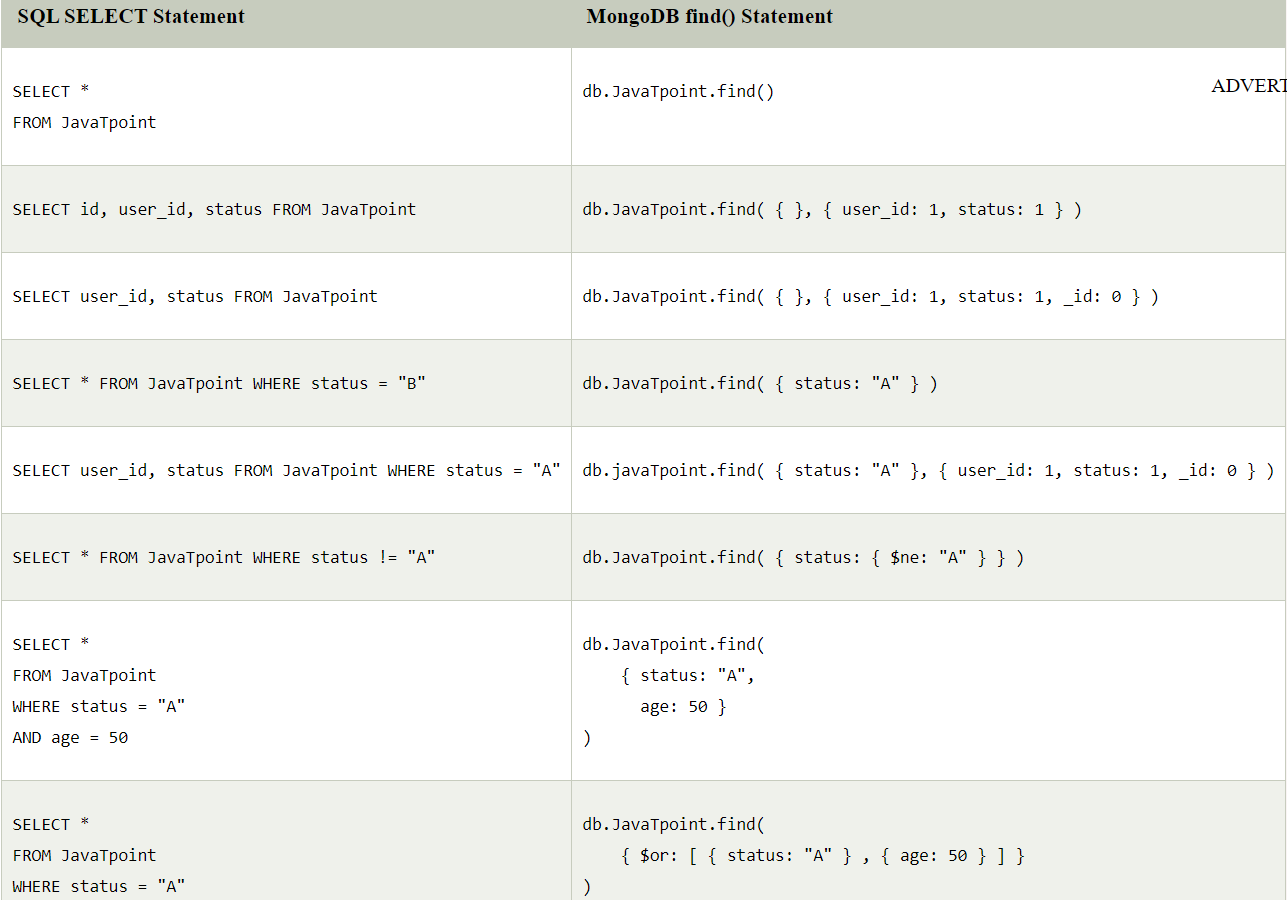
### Create and Alter commands

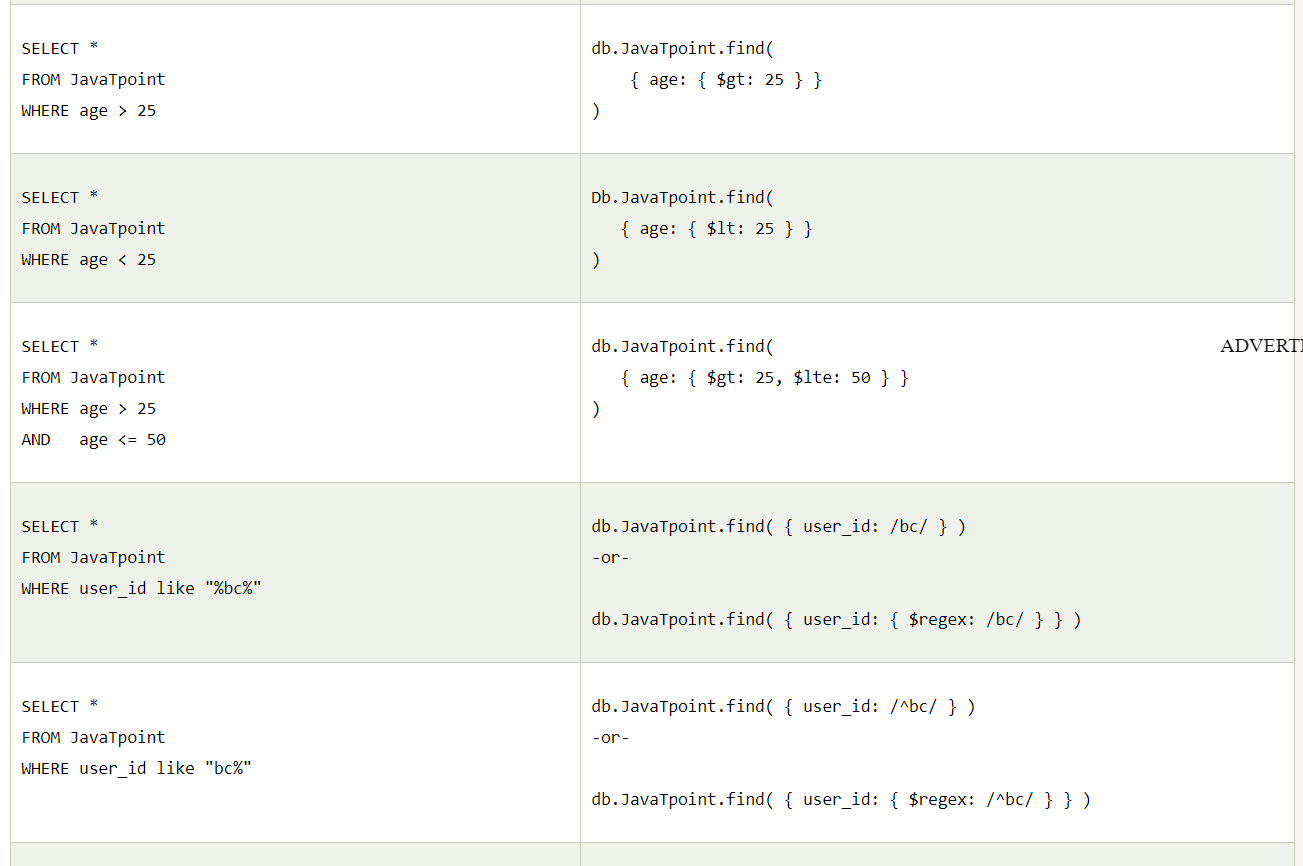


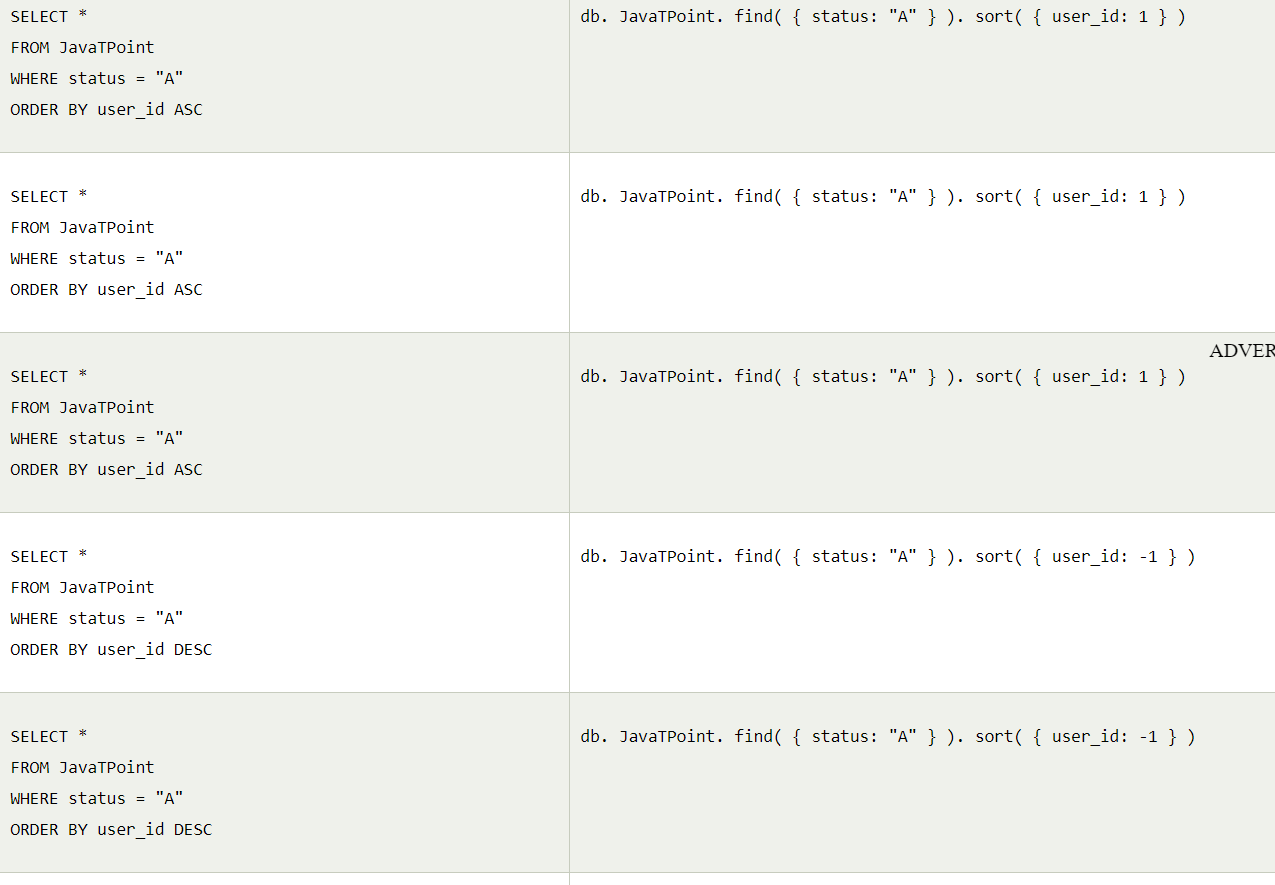


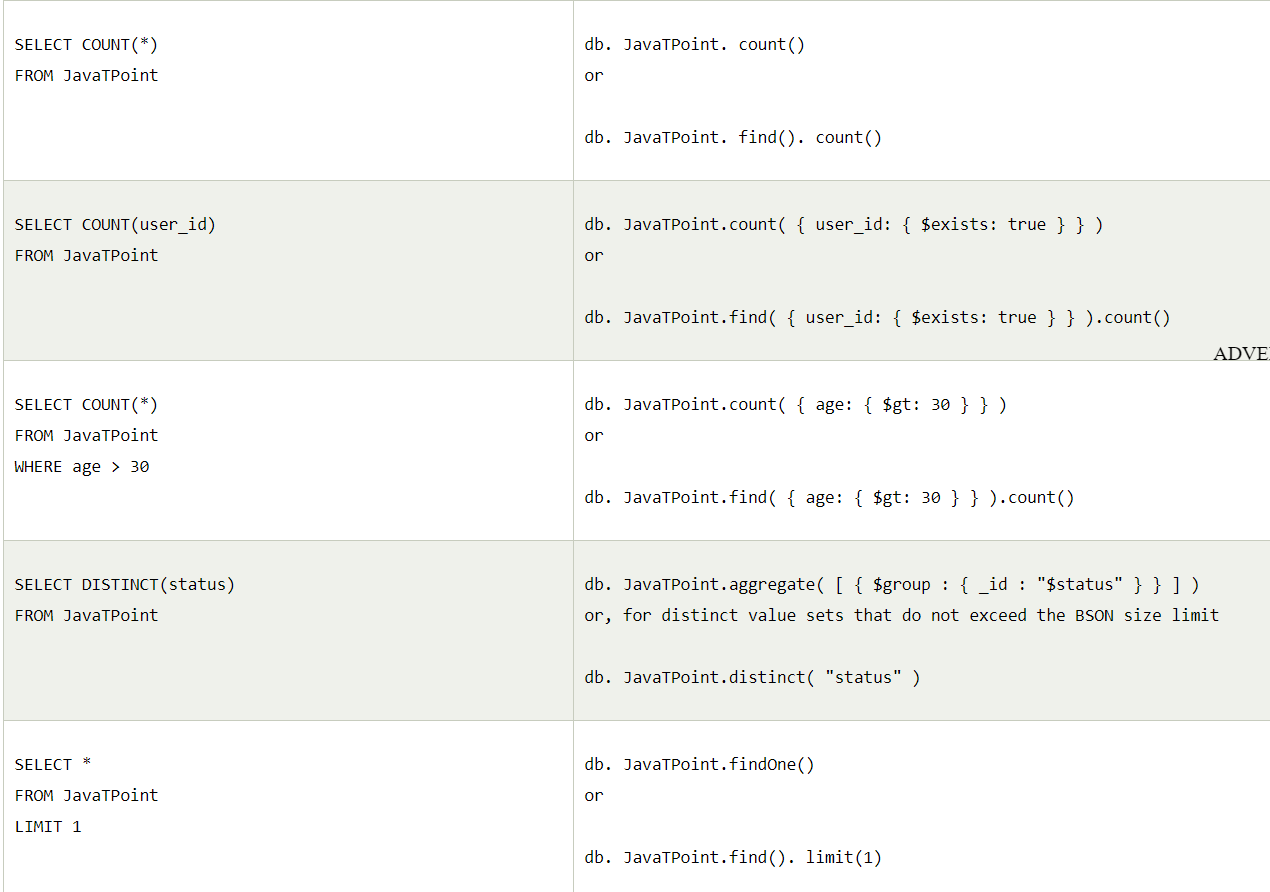


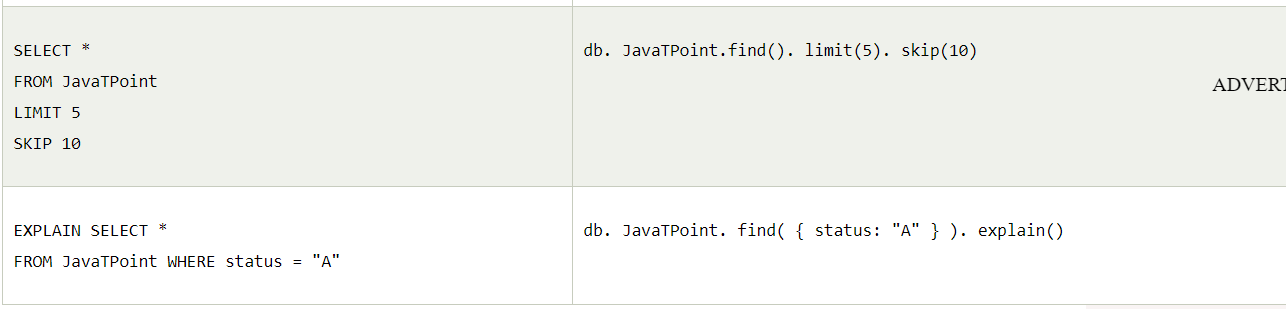
### SQL and Mongo DB Select Command











### SQL and MongoDB Update Statements

